A biologically inspired framework for Self healing Overlay Systems

Sriram Chellappan, Xun Wang and Dong Xuan

Department of Computer and Information Science,

The Ohio State University,

Columbus, OH 43210

{chellapp, wangxu, xuan}@cis.ohio-state.edu

Abstract: In this report, we study the issue of providing self healing behavior to overlay systems following biological principles. Specifically we study the mechanisms of tissue wound healing in human beings and apply the principles to Intermediate Forwarding Overlay systems. We design a protocol that follows from the tissue wound healing mechanisms and discuss preliminary insights we have obtained on its performance under attacks on the Intermediate Forwarding Overlay systems. We believe that significant performance benefits can be obtained using biologically inspired healing mechanisms especially under intensive attacks.

1. Introduction

In simple words, an overlay system consists of a set of nodes where each application (running on a node) establishes connections with other applications and thereby builds a topology among them on top of the existing IP layer. Due to its features of being easily deployed and application-controlled, overlay systems have tremendous scope in the future towards designing newer systems for different applications. Applications of such systems are (1) Content Delivery Networks (CDN) [2], (2) Video conferencing [3] and (2) file sharing [4]. Recent research has extended to scope of overlay systems to provide critical and emergency communication services to the government and military administration [1, 6]. Such intermediate forwarding systems themselves will be the target of attacks by adversaries and hence need to be resilient and 'self-healing' such that they can effectively recover from failures and still maintain performance levels. In this report, we study to how to apply the tissue wound healing principles and mechanisms employed by the human body to make such systems self-healing. The report is organized as follows. In Section 2, we discuss our motivations for this study. We describe the Tissue wound healing mechanism in Section 3. Section 4 discusses our self-healing protocol under attacks. We discuss preliminary observations and scope for future work in Section 5.

2. Motivation

The Intermediate Forwarding System (IFS) typically consists of a set of overlay nodes acting as a communication medium between clients and a critical server. Clients contact the overlay nodes that in turn talk to the server. Depending on the service scope, these systems will be comprised of nodes diverse in terms of geography, processing power, functionality etc.

Considering that the intermediate forwarding systems (IFS) will be used for emergency communication services, they are prone to attacks. Attacks on nodes can have serious consequences in terms of disrupting communication between the clients and the target. We consider two threats to the IFS system, (1) Break-in attacks and that aim to disclose more nodes and maybe eventually the critical server and (2) Congestion Attacks that aim to congest a node in the system effectively preventing it from servicing requests. Nodes disclosed can be targets of future congestion attacks. This naturally leads to designing good defense mechanisms for these systems. It is obvious that under large scale and/ or unexpected attacks, it is an onerous task to maintain performance levels by relying on any manually repair mechanism. It is very important for the system to be self-healing by itself such that attacks effects can be effectively minimized.

In this report we study the design self-healing intermediate forwarding systems, deriving inspiration from many biological organisms that are inherently self-healing. Biological mechanisms have been successfully used in the past for solving many real-world problems [7, 8]. Our approach to address the issue of self-healing IFS systems derives inspiration from the tissue wound healing mechanisms in human beings fundamentally because of the direct similarities in the attack effects in both the cases and the special features of the Intermediate forwarding overlay system that lends itself to apply the tissue healing mechanisms as we discuss later.

3. Tissue Wound Healing Process

Towards realizing self-healing IFS systems, as a first step, we study the tissue healing process in human beings. The following are phases in the healing system when reacting to any injured tissue.

1 **Bleeding phase:** This phase typically is the system's initial reaction to an injury. Depending on the nature of injury, the bleeding may be significant.

2 **Inflammatory Phase:** This phase is a normal and necessary prerequisite to healing. The cells surrounding the affected area immediately react by killing poisonous or dead tissues present and they set up mechanisms to signal and facilitate the successive repair process. To do this, the cells surrounding the affected tissue (such as master cells, platelets and basophiles) send chemical **mediators** out. One result is an increase in the vaso-permeability of the local vessels, which allows increased exudates production such as plasma proteins. This production facilitates the successive healing process. Plasma proteins provide the essential components to generate new tissues.

3 **Proliferation Phase:** This is the first step of the actual repair. New tissues will be generated from plasma proteins to replace the lost ones. Many such tissues are generated and such 'repair tissues' are 'coarse' in the sense that they are not 'perfect' replacements for the lost tissue. They are just a temporary alleviation to replace the lost tissues that can bring the damage effects under control and resume the workability of the affected organ swiftly.

4 **Re-modeling Phase:** The repair tissues are eventually remodeled so that they can exactly replace the lost tissues. This process is neither swift nor highly reactive. This phase is eventually necessary as otherwise the affected organ cannot recover 'fully' and may still show signs of the injury. In order to accomplish this, the organ needs to be in a workable state (at least partially) so that over a period of time the replaced tissue can bring itself to exactly replace the lost tissues by getting itself molded due to operation of

the organ. This phase is typically an instance of physical therapy.

It is easy to see that the effects of the attacks we consider disrupting the IFS system are similar to the effects of a wound in the human body. There is a loss of resources, a detectable change in the systems state and its performance and the need for fast reaction. More importantly, overlay systems naturally have some salient features that lend themselves towards applying such mechanisms for recovery. Overlay networks are flexible with more control at the application layer and addition of new functionality towards enabling recovery merely translates to running new applications (software) on the individual nodes. Addition of resources translates to deployment of more nodes (depending on their availability). Other features of overlay system that we will consider are their diversity, the ease of swapping resources etc.

3. A Self-healing protocol for Intermediate Forwarding systems

Before we describe our self-healing protocol, we will describe our system briefly. Our IFS systems typically consist of nodes residing in a set of intermediate layers through which clients communicate with a server. Nodes currently residing in any of the layers are active nodes. Apart from these, there are spare resources available for replacements. Clients communicate with nodes in the first layer that communicate with nodes in the next layer and so on. Nodes in the last layer communicate with the server. Each layer has a Layer head that necessitates any reaction under attacks. Nodes also maintain connections with a subset of nodes in their layer with the connectivity established by the Layer head. As mentioned before, the attacks we consider are break-in attacks and congestion attacks. Figure 1 shows a description of our IFS system.

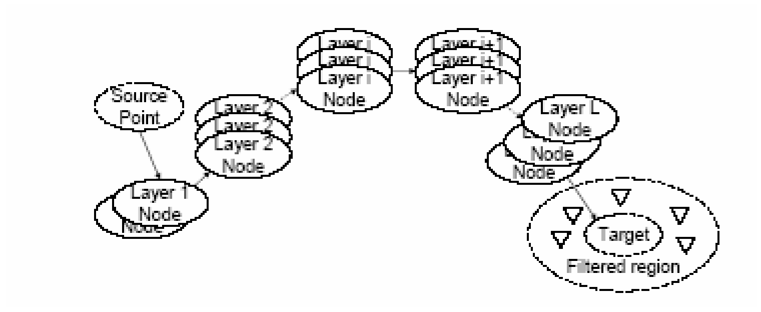

Figure 1. The Intermediate Forwarding Systems Architecture

Our self-healing mechanism follows from the Tissue wound healing mechanism discussed in Section 3. Our approach is to by splitting the healing process into two phases. They are (1) Maintenance Phase and (2) Reaction Phase. The Maintenance phase (Main protocol) is periodically performed by all the nodes while the Reaction phase (Reaction protocol) is performed only by the Layer head of a particular layer. In this section we will describe the protocols and give their pseudo code. Appendix 1 contains the actual protocols.

4.1. The Main Protocol

This protocol performs the maintenance. Each node in the IFS periodically checks with its previous layer nodes for their availability. We assume that there is a mechanism to identify the set of nodes that are congested or broken into (typically, we call such a node as compromised). Once a node detects that its previous layer nodes are compromised it forwards the information to its neighbors that eventually reaches the layer head of that layer. The layer head upon receiving this information takes appropriate action in terms of just replacing the compromised nodes under mild attack intensities or take stronger action under heavier intensities. The following is the pseudo code for the protocol.

Pseudo code for the Main Protocol:

```
// Performed by each node in the system except first layer nodes for maintenance.

Module Main ()
do
```

```
        nbr = Get neighbors in the previous layer
        nbr_cong = Get congested neighbors in previous layer
        nbr_broken = Get brokeninto neighbors in previous layer
        If (0 < nbr_cong) OR (0 < nbr_broken)
            inform other neighbors of the list of compromised nodes
        End If
        If LayerHead
            nbr_total_cong = get all congested nodes in previous layer
            nbr_total_broken = get all boken-into nodes in previous layer
            If not contacted by another layer // to assist in recovery
                Call module react // to start the reaction process
            End If
        End If
 While (true)
End Module Main ()
```

4.2. The Reaction Protocol

Once the layer head obtains the set of compromised nodes in the next layer, it decides what sort of reaction to take. This reaction can be to replace newer nodes, merge the layers together or split them. Such a reaction is instinctive in the sense that this may not be an optimum solution. Rather it is a temporary reaction to the attack. The layer head also periodically checks to see if any further optimization can be done even during the absence of attacks. Such operations can be performed only if the layer head can really determine the Global optimum (at least within a certain bound). We are in the process of designing such functions for the same.

Pseudo code for the Reaction Protocol:

```
// Performed at each layer by Layer head only
Input: The set of congested nodes and broken-in nodes in the previous layer
Module Reaction (set a, set b)
Set Flag =1 // To indicate reaction being performed
if (nbr_total_cong = 0) OR (nbr_total_broken = 0) // No attack
            Goto Stage:
endif
Find q nodes to replace the set a and b
```

```
If found
        Replace the set of congested and broken-in nodes and connections
Else // System is in a state of alert
Stage:  Call Function to determine nature of reaction and resources needed
        If merge needs to be done
            Contact previous layer head to merge
            Reestablish routing tables and connections
        End if
        If split needs to be done
            Remove selected nodes from layer
            Forward to layer before which a layer needs to be added
            Reestablish connections and Routing tables
        End if
End if
Set Flag = 0.
End Module Reaction ()
```

5. Discussion and Future Work

We have obtained significant insights into the performance of the IFS system in [5]. There we discovered good properties of the IFS system that is static without dynamics. We clearly studied the effects of three properties of IFS systems, namely (1) node density (number of nodes in each layer), (2) connectivity degree across the layers and (3) number of layers. We will leverage our findings there in using these three features in our self-healing protocol. The following are introductory conclusions that we have obtained in the design of self-healing IFS systems. Under high intensities of Break-in attacks, we should dynamically increase the number of layers in the system to make it harder to disclose the target and also reduce the number of nodes disclosed. Under congestion attacks, we should merge layers together primarily in order to increase the number of connections across layers. Such a step may not be optimum in performance because the primary goal is to restore communication and reduce attack effects. Eventually the system will 'remodel' itself to better states. We are in the process actually implementing the protocol and trying to obtain clearer insights into self-healing IFS systems. Another direction of our current work is in designing biologically aware IFS system structures. By this we wish to employ proactive protection mechanisms to our IFS system by building the structure based on principles of biological mechanisms. These involve virus defense structures, structure enabling localization of attacks etc. It is also our belief that this study will help eventually us in designing biologically inspired self-healing protocols for other classes of overlay systems.

References

1. A. Keromytis, V. Misra, and D. Rubenstein, "SOS: Secure Overlay Services," in Proceedings of ACM SIGCOMM'02, (Pittsburgh, PA), August 2002.
2. J. Byers, J. Considine, M. Mitzenmacher, and S. Rost. “Informed content delivery across adaptive overlay networks,” In Proceedings of ACM SIGCOMM 2002.
3. Yang-Hua Chu, Sanjay Rao, and Hui Zhang. “A case for end system multicast,” In Proceedings of ACM Sigmetrics, Santa Clara, CA, 2000. Bio-networking
4. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. “Chord: A scalable peer-to-peer lookup service for Internet applications,” Technical Report TR-819, MIT, March 2001.
5. D. Xuan. S. Chellappan, X. Wang and S. Wang, Analyzing Secure Overlay Services Architectures under intelligent DDoS attack, to appear in Proc. of IEEE International Conference on Distributed Computing Systems (ICDCS) 2004.
6. David G. Andersen, Mayday: Distributed Filtering for Internet Services, 4th Usenix Symposium on Internet Technologies and Systems, Seattle, Washington, March 2003.
7. Jeffrey Kephart, A biologically Inspired Immune System for Computers, Artificial Life IV, R. Brooks and P. Maes eds., MIT Press, 1994.
8. Michael Wang, Tatsuya Suda: The Bio-Networking Architecture: A Biologically Inspired Approach to the Design of Scalable, Adaptive, and Survivable/Available Network Applications, Symposium on Applications and the Internet, San Diego 2001.

Appendix 1.

We describe the main and reaction protocols and the corresponding function and message handlers here.

1. The Main Protocol

Input: No Input

```
Module Main ()
// Performed by all nodes except ones at the First Layer for maintenance
// Periodically each node checks with its neighbors in previous (i-1) layer.
do
        ni.original_nbr_set = func_get_nbrs()
        ni.congested_neighbor<- func_get_prev_cong_neighbor_list()
        ni.brokeninto_neighbor<- func_get_prev_brokeninto_neighbor_list()
        If (0 < ni.congested_neighbor ) OR (0 < ni.brokeninto_neighbor)
            send message neighbor_down(original_nbrs,congested nodes, broken-into nodes) to
        neighbors
        // The above message will eventually reach Lhi
                If LayerHead then
                      prev_congested_nodes = func_get_congested_list ()
                      prev_brokeninto_nodes = func_get_brokeninto_list ()
                      If Merge_Flag ==0 then
                           call module react(prev_congested_nodes, prev_brokeninto_nodes)
 While(true)

End Module Main ()
```

2. The Reaction Protocol

Input: Set of congested and broken-in nodes

```
Module reaction (set a, set b)
// Layer head i does this and replaces (i-1) layer nodes
Set Flag = 1 // to indicate some operation is currently being done.
if (a=0) and (b=0) // no attacks for remodelling purposes
  GoTo Stage:
end
// Obtain q new nodes to replace set a and b
        q <- func_get_new_nodes(|a U b|)
        if q = (|a U b|) then // Put the q nodes in the overlay
        lhi. prev_Layer_Working_Set <- func_get_all_prev_list() // stored in the db. get K
        if (Lh(i-1) € (a or b))
```

```
                lh(i-1) <- func_choose_layer_head(q U lhi.current_prev_layer_list)
            send message_arrange_Peers to lh(i-1)
             //lh(i-1).peer_list <- func_arrange_peers(q U W(i-1))
             lhi. prev_Layer_Working_Set<- func_finalize_connections(lh(i-1).peer_list)
             send message restablish_broken_conn_with_lower_layer (set q) // to Lh(i-1) to
             end
             end
if (all not replaced) then
 Stage:    Lhi.new_data_structure = func_advice (int a, int b) // DataStructure contains all info needed to react
              If Lhi.new_data_structure.merge = True
              Lhi.peer_list<- func_get_all_peer_list() // get my nbr's and so on
              lhi.current_prev_layer_list<- func_get_all_prev_list()
              if Lhi.prev_layer_working_set==o then
                       send message partition_resolve() to upper layer
              Lhi.prev_prev_layer_working_set<-send Merge_Request message to Lh(i-1)
              Lhi.prev_prev_working_set <-send message_get_me_Peer_list() to Lh(i-1)
              send message relinquish_head() to node that was previously Lh
              lh(i-1).finalizeconnecitons(Lh(i-1).prev_prev_layer_working_set) // i<- i-1
              Forward message update_conn(Lh(i-1).peer_list)to Lh(i) // one layer up
   End
  If Lhi.new_data_structure.split = True
              Lhi.peer_list<- func_get_all_peer_list() // get my nbr's and so on
               lhi..current_prev_layer_list<- func_get_all_prev_list()
              Lhi.newdatastructure.split_set <- func_split_advice(lhi.peer_list)
              //split_set contains set nodes to remove
              // nodes removed are forwarded to some fixed layer say B <=L.
              Lhi.peerlist <- arrange_peers(Lhi.peer_list - nodes removed) //
              Lhi.func_finalize_connections(lhi.current_prev_layer_list)
              Forward message update(Lhi.peer_list to Lh(i+1))
              send add_layer_msg (set R, B) to Lh(i+1)
  end
Set Flag =0.
End Module Reaction ()
```

Output: A new structure

3. Function Handlers:

func_choose_layer_head(set ni)

```
  {
    // Among the nodes in the input, choose the most powerful one P
    Return P.
    }
```

func_arrange_peers(set ni)

```
  {
   // The purpose of this function is for the Layer head ask each node to maintian
       information about only some nodes among its peers. This is to ensure that each node
       does not know all nodes in the same layer, while the layerhead can still
       commmunicate with all nodes in the layer.
    // At the end, each node (including Layer head) will know a subset of nodes in its layer
    Return Layerhead's contacts in its layer
    }
```

func_finalize_connections(Set q)

```
{
  if |q| < d(i-1) // obtained from structure property
                find q-d(i-1) more nodes.
                q = q Union d(i-1)
  // Set up connections for the nodes and inform them of their lower_nbr_list depending
     on dml(i) (obtained from Property)
  // Set up connections for each of the q for their upper_nbr_list depending on dmu(i-1)
     (obtained from Property)
if message received from other nodes
   func_arrange_peers
else
send message  arrange_peers(q) to Lh(i-1)
  //Return q. // q=ni for the previous layer (i-1)
}
```

func_get_nbrs()

```
{
          // get nodeid's in routing table in previous layer
          Return Prev_nbrs
}
```

```
func_get_prev_cong_neighbor_list()
{
        // get nodeid's is routing table in previous layer that are congested
        // Obtained thru significant delay is 'Hello' messages
        Return neighbors that are congested
}

func_get_prev_brokeninto_neighbor_list()
{
        // get nodeid's is routing table in previous layer that are broken into
        // Obtained thru decoy nodes mechanism etc
        Return neighbors that are disclosed or have broken-into
}

func_get_congested_list ()
{
 set a=  func_get_prev_cong_neighbor_list()

send message get_all_congested(set a)
  }

func_get_brokeninto_list ()
{
   set a=  func_get_prev_brokeninto_neighbor_list()
   send message get_all_brokeninto(set a)
  }

func_advice(set a, set b)
{
// Determine the effect of Set a and Set b
Return desired properties // Return What to do
}

func_split_advice(set a)
{
// Determine the effect of Set a
Return set b // set of nodes to split
}
```

func_get_all_prev_list()

```
{
        // get nodeid's in routing table in previous layer and
        a  = func_get_nbrs()
        send message get_all_prev_list(my ID, set a) to all nbrs
        Return set p // all nodes in prev layer
}
```

func_get_all_peer_list()

```
{
        // get nodeid's in routing table in previous layer and
        a  = func_get_nbrs()
        send message get_all_prev_list(my ID, set a) to all nbrs
        Return set p // all nodes in prev layer
}
```

4. Message Handlers:

message neighbor_down (k,set a, set b)

```
{
congested nodes =  a U  func_get_prev_cong_neighbor_list()
broken-into nodes = b U  func_get_prev_brokeninto_neighbor_list()
total_nbrs = k U func_get_nbrs()
        send message neighbor_down(K, congested nodes, broken-into nodes) to neighbors

 // Hanndled only by a LayerHead
       If K = Null
            send partitionresolve message to upper Layer
       // Store K in database temporarily
       Call func_react (congested nodes, broken-into nodes)
}
```

message Partition_Resolve()

```
{
// handled by layer heads
if authorized to handle it then
  send partition recovery message to first layer Lhi // this layer has the info
else
  forward the meggage to upper Layer Head
end
}
```

```
message Partition_Recovery()
{
// Handled by Lhi
Forward the message to upper Lhi.
if Partition is detected then decrypt the message to read ID of nodes in the message
          send message my_current_set_to_resolve_partition to Lh in the ID’s
end
}

message my_current_set_to_resolve_partition (current_set)
{
// handled by Lhi
  Read the current_set from the message
  finalize_connections(current_set)
}


message add_layer (set R, int B)
{
// handled by layer_head i
if (i<>B)
             Forward to next lh(i+1)
elseif (i==B)
                //Read the message and obtain Id’s of the nodes
                Lhi.chooselayerhead(set R) // R = layerhead in i-1
                Lhi.finalizeconnections(R)
                   a = func_get_all_prev_list()
                   send message update_conn(set a) to Lh(i-1)
                // Lh(i-1).arrange_peers
}

message get_all_brokeninto(set a)
{
set a = a U func_get_prev_brokeninto_neighbor_list()
send message get_all_brokeninto(set a) to all nbrs
If LH then stop forwarding
    Return set a

}
```

message get_all_congested(set a)

```
{
set a = a U func_get_prev_cong_neighbor_list()
send message get_all_congested(set a) to all nbrs
If LH then stop forwarding
    Return set a

}
```

message get_all_prev_list(ID, set a)

```
{
set a = a U func_get_nbrs()
ID = myID U ID
send message get_all_brokeninto(ID, set a) to all nbrs
If LH then stop forwarding
    Return set a
}
```

message restablish_broken_conn_with_lower_layer (set q)

```
{
// Hanlded by LayerHead i
    func_arrange_Peers(set q)
    send message get_me_Peer_list to prev Layerhead (i-1)
    func_finalize_connnections
}
```

message get_me_Peer_list()

```
{
// handled by layer head i
send message forward_Id's (my id) to nbrs
}
```

message forward_Id's (id)

```
{
// handled by each node
id = idU myid
send message_forward_Id's (myid) to nbrs
if Layerhead
   return set myid
}
```

```
message  arrange_peers(set q)
{
// handled by Lhi
func_arrange_peers(set q)
}

message update_conn(set a)
{
func_finalize_connections(set a)
}

message_start_maintenance()
{
module maintenance()
}

message_relinquish_head()
{
// LayerHead i relinquishes its role as a head completely
}

message_merge_request()
{
Merge_Flag =1
Stage:
        Check Flag
        if Flag <> 1 then
            Lh(i).next_layer_working_Set <- func_get_all_prev_list() // done by Layer Head i-1
        else
            Goto Stage
        end
}
```